FIG. 1

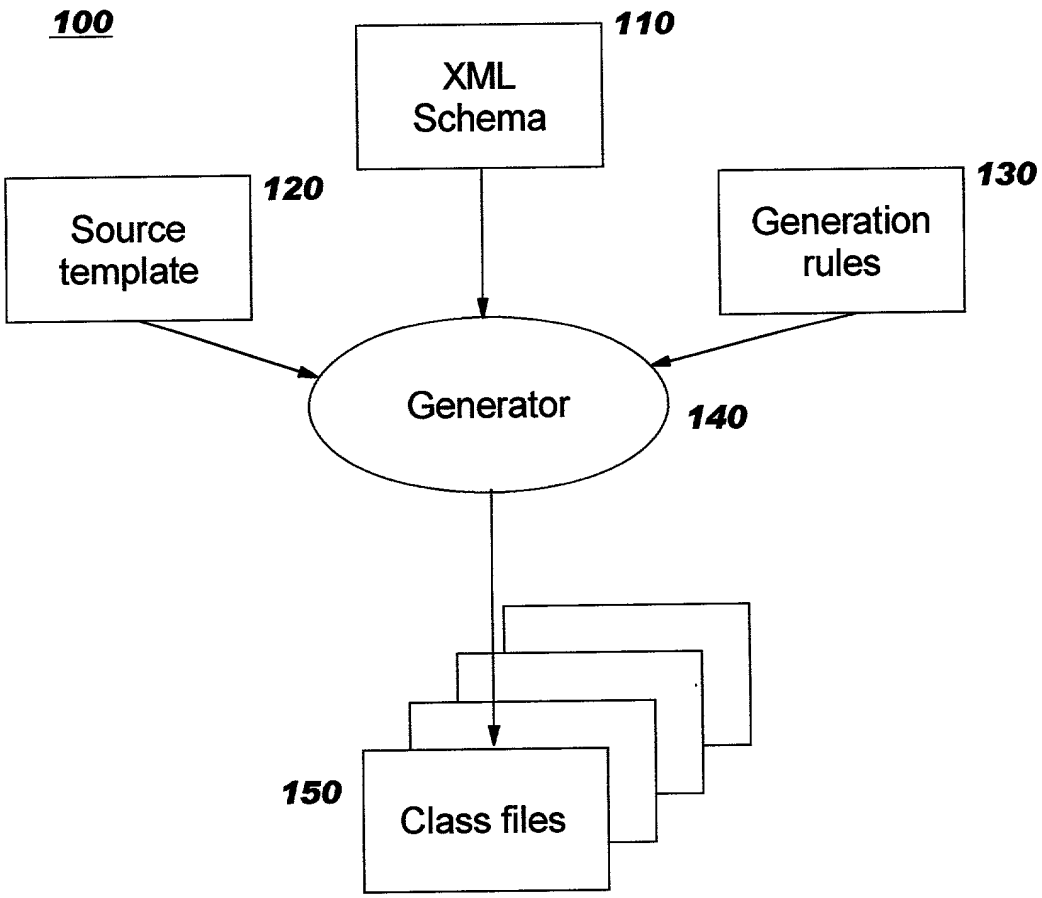

100
XML Schema
110
Source template
120
Generation rules
130
Generator
140
150
Class files

FIG. 2

200

```
<element name = "discoveryURL">                                    210
    <type source = "string" content = "textOnly">                  220
        <attribute name = "useType" minOccurs = "1" type = "string"/>
    </type>                                                        230
</element>
```

FIG. 3

300

```
#-------------------------------------------
# Set directory to output generated files to
#-------------------------------------------
outputDirectory = .                                                310

classJavadoc.0  = ' * <p><b>General information:</b><p>            320

FindQualifier.classVariableDeclare.0 = '   /** Valid values */     330
FindQualifier.classVariableDeclare.1= '   public static final String exactNameMatch =
     "exactNameMatch";                                             340

DispositionReport.dontGenerate = true                              350

DiscoveryURL.packageName    = com.ibm.util                         360
# -----------------------------------------
```

FIG. 4A

400

```
package com.ibm.util;                                          405

/** <p><b>General information:</b><p> */                       410
public class DiscoveryURL {                                    415
  String text = null;                                          420
  String useType = null;                                       425

  public DiscoveryURL() {
  }                                                            430

  /**
   * Construct the object with required fields.                435
   *
   * @param value     String value                             440
   * @param useType   String                                   445
   */
  public DiscoveryURL(String value,
        String useType) {
    setText(value);                                            450
    this.useType = useType;
  }

  /**                                                          455
   * Construct the object from a DOM tree.  Used by
   * UDDIProxy to construct an object from a received UDDI
   * message.
   *
   * @param base   Element with name appropriate for this class.
   *
   * @exception UDDIException Thrown if DOM tree contains a SOAP fault
   *  or a disposition report indicating a UDDI error.
   */
  public DiscoveryURL(Element base) {                          460
    // Check if its a fault. Throw exception if it is
    super(base);                                               465
    text = getText(base);
    useType = base.getAttribute("useType");
  }
```

FIG. 4B

470

```
public void setText(String s) {
   text = s;
}

public String getText() {
   return text;
}
```

475

```
public void setUseType(String s) {
   useType = s;
}

public String getUseType() {
   return useType;
}
```

480

```
/**
* Save an object to the DOM tree. Used to serialize an object
* to a DOM tree, usually to send a UDDI message.
*
* <BR>Used by UDDIProxy.
*
* @param parent Object will serialize as a child element under the
 *  passed in parent element.
 */
public void saveToXML(Element parent) {
   base = parent.getOwnerDocument().createElement(UDDI_TAG);
   // Save attributes
   if (text!=null) {
      base.appendChild(parent.getOwnerDocument().createTextNode(text));
   }
   if (useType!=null) {
      base.setAttribute("useType", useType);
   }
   parent.appendChild(base);
 }
}
```

FIG. 5

500

```
// Create and interact with classes to create message                    505
DiscoveryURL durl = new DiscoveryURL("url_value", "usetype_value");     510
durl.setXXX   // set any additional attributes. This particular class has none.
                                                                        515
// Create element to serialize object to                                520
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder docBuilder = factory.newDocumentBuilder();
Element base = docBuilder.newDocument().createElement("tmp");

// Invoke generated serialization method                                525
durl.saveToXML(base);

// Convert DOM to format used by messaging interface                    530
String message = DOMWriter.nodeToString(el)

// Invoke desired mechanism to send/receive an XML message              535
transport.send(message);
String response = transport.receive();

// Convert response back to generated object form                       540
Element responseEl = parse(response);
DiscoveryURL response = new DiscoveryURL(responseEl);

// Can interact with object form of the message                         545
System.out.println(response.getText());
System.out.println(response.getUseType());
```

FIG. 6

600

```
<element name = "businessEntity">
    <type content = "elementOnly">                    605
      <annotation>                                    610
        <appInfo>
              Primary Data type: Describes an instance of
              a business or business unit.
        </appInfo>
      </annotation>
      <group order = "seq">
        <element ref = "discoveryURLs" minOccurs = "0" maxOccurs = "1"/>
        <element ref = "name"/>                       615
        <element ref = "description" minOccurs = "0" maxOccurs = "*"/>
        <element ref = "contacts" minOccurs = "0" maxOccurs = "1"/>
        <element ref = "businessServices" minOccurs = "0" maxOccurs = "1"/>
        <element ref = "identifierBag" minOccurs = "0" maxOccurs = "1"/>
        <element ref = "categoryBag" minOccurs = "0" maxOccurs = "1"/>
      </group>
      <attribute name = "businessKey" minOccurs = "1" type = "string"/>
      <attribute name = "operator" type = "string"/>
      <attribute name = "authorizedName" type = "string"/>
    </type>
  </element>
```

700

FIG. 7A

```
/*
 * The source code contained herein is licensed under the IBM Public License
 * Version 1.0, which has been approved by the Open Source Initiative.
 * Copyright (C) 2001, International Business Machines Corporation
 * All Rights Reserved.                                              702
 *
 */

package %packageName%;        704

import java.util.Vector;        706
import org.w3c.dom.*;
import com.ibm.uddi.*;
import com.ibm.uddi.datatype.*;
import com.ibm.uddi.datatype.binding.*;
import com.ibm.uddi.datatype.business.*;
import com.ibm.uddi.datatype.service.*;
import com.ibm.uddi.datatype.tmodel.*;
import com.ibm.uddi.request.*;
import com.ibm.uddi.response.*;
inport com.ibm.ussi.util.*;

/**                                        708
 * <p><b>General information:</b><p>
 *
 * This class represents an element within the UDDI version 1.0 schema.
 * This class contains the following types of methods:<ul>
 *
 *  <li>A constructor that passes the required fields.
 *  <li>A Constructor that will instantiate the object from an XML DOM element
 *       that is the appropriate element for this object.
 *  <li>Get/set methods for each attribute that this element can contain.
 *  <li>For sets of attributes, a get/setVector method is provided.
 *  <li>A SaveToXML method that serializes this class within a passed in
 *       element.
 * </ul>
```

FIG. 7B

```
 * Typically, this class is used to construct parameters for, or interpret
 * responses from, methods in the UDDIProxy class.
 *
 * <p><b>Element description:</b><p>
 *
%foreach%annotation%                                    720
 * %annotation%                                         722
%end%
 *
 * <p>
 *
 * @author David Melgar (dmelgar@us.ibm.com)
 */
public class %ElementName% extends UDDIElement {
  public static final String UDDI_TAG = "%elementName%";

  protected Element base = null;

%ifText%
  String text = null;
%end%
%forEach%Attribute%
  String %attribute% = null;
%end%A
%forEach%Child%
  %Child% %child% = null;
%end%
%forEach%ChildCollection%
  // Bector of %Child% objects
  Vector %child% = new Vector();
%end%
```

FIG. 7C

```
/**
 * Default constructor.
 * Avoid using the default constructor for validation. It does not validate
 * required fields. Instead, use the required fields constructor to perform
 * validation.
 */
public %ElementName%() {
}

/**
 * Construct the object with required fields.                    746
 *
%ifText%
 * @param value  String value                                    748
%end%
%forEach%required%attribute%                                    750
 * @param %attribute%  String
%end%
%forEach%Child%
%end%
%forEach%ChildCollection%
%end%
 */
public %ElementName%(%forEach%required%,%end%) {                752
%ifText%
    setText(value);                                             754
%end%
%forEach%required%Attribute%                                    756
    this.%attribute% = %attribute%;
%end%
%forEach%required%TextOnlyChild%
    this.%child% = new %Child%( %child% );
}
```

FIG. 7D

```
/**
 * Construct the object from a DOM tree. Used by
 * UDDIProxy to construct an object from a received UDDI
 * message.
 *
 * @param base   Element with name appropriate for this class.
 *
 * @exception UDDIException
 *              Thrown if DOM tree contains a SOAP fault or
 *              disposition report indicating a UDDI error.
 */
public %ElementName%(Element base) throws UDDIException {
  // Check if it is a fault. Throws an exception if it is.
  super(base);
%ifText%
  text = getText(base);
%end%
%forEach%Attribute%
  %attribute% = base.getAttribute("%attribute%");
%end%
  NodeList nl = null;
%forEach%Child%
  nl = getChildElementsByTagName(base, %Child%.UDDI_TAG);
  if (nl.getLength() > 0) {
    %Child% = new %Child%((Element)nl.item(0));
  }
%end%
%forEach%ChildCollection%
  nl = getChildElementsByTagName(base, %Child%.UDDI_TAG);
  for (int i=0; i < nl.getLength(); i++) {
    %child%.addElement(new %Child%((Element)nl.item(i)));
  }
}
```

760

FIG. 7E

```
%ifText%                                          780
  public void setText(String s) {
    text = s;
  }

  public String getText() {                       782
    return text;
  }
%end%

%forEach%Attribute%                               784
  public void set%attribute%(String s) {
    %attribute% = s;
  }

                                                  786
  public String get%attribute%() {
    return %attribute%;
  }
%end%
```

FIG. 7F

```
/**
 * Save an object to the DOM tree. Used to serialize an object
 * to a DOM tree, usually to send a UDDI message.
 *
 * <BR>Used by UDDIProxy.
 *
 * @param parent Object will serialize as a child element under the
 *  passed in parent element.
 */
public void saveToXML(Element parent) {                    788
  base = parent.getOwnerDocument().createElement(UDDI_TAG);
  // Save attributes.
%ifText%
  if (text!=null) {                                        790
    base.appendChild(parent.getOwnerDocument().createTextNode(text));
  }
%end%
%forEach%Attribute%
  if (%attribute%!=null) {                                 792
    base.setAttribute("%attribute%", %attribute%);
  }
%end%                                                      794
  parent.appendChild(base);
 }
}
```

FIG. 8A

800

```
/*
 * The source code contained herein is licensed under the IBM Public License
 * Version 1.0, which has been approved by the Open Source Initiative.
 * Copyright (C) 2001, International Business Machines Corporation
 * All Rights Reserved.
 *
 */

package com.ibm.uddi.datatype.business;
import java.util.Vector;
import org.w3c.dom.*;
import com.ibm.uddi.*;
import com.ibm.uddi.datatype.*;
import com.ibm.uddi.datatype.binding.*;
import com.ibm.uddi.datatype.business.*;
import com.ibm.uddi.datatype.service.*;
import com.ibm.uddi.datatype.tmodel.*;
import com.ibm.uddi.request.*;
import com.ibm.uddi.response.*;
import com.ibm.uddi.util.*;

/**
 * <p><b>General information:</b><p>          805
 *
 * This class represents an element within the UDDI version 1.0 schema.
 * This class contains the following types of methods:<ul>
 *
 *   <li>A constructor that passes the required fields.
 *   <li>A Constructor that will instantiate the object from an XML DOM element
 *       that is the appropriate element for this object.
 *   <li>Get/set methods for each attribute that this element can contain.
 *   <li>For sets of attributes, a get/setVector method is provided.
 *   <li>A SaveToXML method that serializes this class within a passed in
 *       element.
 * </ul>
```

FIG. 8B

```
* Typically this class is used to construct parameters for, or interpret
* responses from methods in the UDDIProxy class.
*
* <p><b>Element description:</b><p>
* Primary Data type: Describes an instance of a business or business unit.
*                                                          810
* <p>
*
* @author David Melgar
*/
public class BusinessEntity extends UDDIElement {
  public static final String UDDI_TAG = "businessEntity";

  protected Element base = null;



  String businessKey = null;                    815
  String operator = null;
  String authorizedName = null;
  DiscoveryURLs discoveryURLs = null;
  Name name = null;
  Contacts contacts = null;
  BusinessServices businessServices = null;
  IdentifierBag identifierBag = null;
  CategoryBag categoryBag = null;
  // Vector of Description objects
  Vector description = new Vector();          820
```

FIG. 8C

```
/**
 * Default constructor.
 * Avoid using the default constructor for validation. It does not validate
 * required fields. Instead, use the required fields constructor to perform
 * validation.
 */
public BusinessEntity() {
}

/**
 * Construct the object with required fields.
 *
 * @param businessKey String
 * @param name   String
 */
public BusinessEntity(String businessKey,
        String name) {
  this.businessKey = businessKey;
  this.name = new Name( name );
}

/**
 * Construct the object from a DOM tree. Used by
 * UDDIProxy to construct object from a received UDDI
 * message.
 *
 * @param base   Element with the name appropriate for this class.
 *
 * @exception UDDIException
 *              Thrown if DOM tree contains a SOAP fault or
 *              disposition report indicating a UDDI error.
 */
```

FIG. 8D

```
public BusinessEntity(Element base) throws UDDIException {
  // Check if its a fault. Throw exception if it is.          825
  super(base);
  businessKey = base.getAttribute("businessKey");
  operator = base.getAttribute("operator");
  authorizedName = base.getAttribute("authorizedName");
  NodeList nl = null;
  nl = getChildElementsByTagName(base, DiscoveryURLs.UDDI_TAG);
  if (nl.getLength() > 0) {
    discoveryURLs = new DiscoveryURLs((Element)nl.item(0));
  }
  nl = getChildElementsByTagName(base, Name.UDDI_TAG);
  if (nl.getLength() > 0) {
    name = new Name((Element)nl.item(0));
  }
  nl = getChildElementsByTagName(base, Contacts.UDDI_TAG);
  if (nl.getLength() > 0) {
    contacts = new Contacts((Element)nl.item(0));
  }
  nl = getChildElementsByTagName(base, BusinessServices.UDDI_TAG);
  if (nl.getLength() > 0) {
    businessServices = new BusinessServices((Element)nl.item(0));
  }
  nl = getChildElementsByTagName(base, IdentifierBag.UDDI_TAG);
  if (nl.getLength() > 0) {
    identifierBag = new IdentifierBag((Element)nl.item(0));
  }
  nl = getChildElementsByTagName(base, CategoryBag.UDDI_TAG);
  if (nl.getLength() > 0) {
    categoryBag = new CategoryBag((Element)nl.item(0));
  }
  nl = getChildElementsByTagName(base, Description.UDDI_TAG);
  for (int i=0; i < nl.getLength(); i++) {
    description.addElement(new Description((Element)nl.item(i)));
  }
}
```

FIG. 8E

```
public void setBusinessKey(String s) {
  businessKey = s;
}

public void setOperator(String s) {
  operator = s;
}

public void setAuthorizedName(String s) {
  authorizedName = s;
}

public void setDiscoveryURLs(DiscoveryURLs s) {
  discoveryURLs = s;
}

public void setName(Name s) {
  name = s;
}
public void setName(String s) {
  name = new Name();
  name.setText(s);
}

public void setContacts(Contacts s) {
  contacts = s;
}

public void setBusinessServices(BusinessServices s) {
  businessServices = s;
}

public void setIdentifierBag(IdentifierBag s) {
  identifierBag = s;
}
```

FIG. 8F

```
public void setCategoryBag(CategoryBag s) {
  categoryBag = s;
}

/**
 * Set description vector                    830
 *
 * @param s Vector of <I>Description</I> objects.
 */
public void setDescriptionVector(Vector s) {
  description = s;
}

/**
 * Set default (english) description string
 *
 * @param s String
 */
public void setDefaultDescriptionString(String s) {          835
  if (description.size() > 0) {
    description.setElementAt(new Description(s), 0);
  } else {
    description.addElement(new Description(s));
  }
}

public String getBusinessKey() {
  return businessKey;
}

public String getOperator() {
  return operator;
}
```

FIG. 8G

```
public String getAuthorizedName() {
  return authorizedName;
}


public DiscoveryURLs getDiscoveryURLs() {
  return discoveryURLs;
}


public Name getName() {
  return name;
}

public String getNameString() {
  return name.getText();
}

public Contacts getContacts() {
  return contacts;
}


public BusinessServices getBusinessServices() {
  return businessServices;
}


public IdentifierBag getIdentifierBag() {
  return identifierBag;
}


public CategoryBag getCategoryBag() {
  return categoryBag;
}
```

FIG. 8H

```
/**
 * Get description
 *
 * @return s     Vector of <I>Description</I> objects.
 */
public Vector getDescriptionVector() {
  return description;
}

/**
 * Get default description string
 *
 * @return s     String
 */
public String getDefaultDescriptionString() {
  if ((description).size() > 0) {
    Description t = (Description)description.elementAt(0);
    return t.getText();
  } else {
    return null;
  }
}


/**                                                    840
 * Save an object to the DOM tree. Used to serialize an object
 * to a DOM tree, usually to send a UDDI message.
 *
 * <BR>Used by UDDIProxy.
 *
 * @param parent Object will serialize as a child element under the
 *  passed in parent element.
 */
```

FIG. 8I

```
public void saveToXML(Element parent) {
  base = parent.getOwnerDocument().createElement(UDDI_TAG);
  // Save attributes
  if (businessKey!=null) {
    base.setAttribute("businessKey", businessKey);
  }
  if (operator!=null) {
    base.setAttribute("operator", operator);
  }
  if (authorizedName!=null) {
    base.setAttribute("authorizedName", authorizedName);
  }
  if (discoveryURLs!=null) {
    discoveryURLs.saveToXML(base);
  }
  if (name!=null) {
    name.saveToXML(base);
  }
  for (int i=0; i < description.size(); i++) {
    ((Description)(description.elementAt(i))).saveToXML(base);
  }
  if (contacts!=null) {
    contacts.saveToXML(base);
  }
  if (businessServices!=null) {
    businessServices.saveToXML(base);
  }
  if (identifierBag!=null) {
    identifierBag.saveToXML(base);
  }
  if (categoryBag!=null) {
    categoryBag.saveToXML(base);
  }
  parent.appendChild(base);
 }
}
```

845

850

855

FIG. 9A

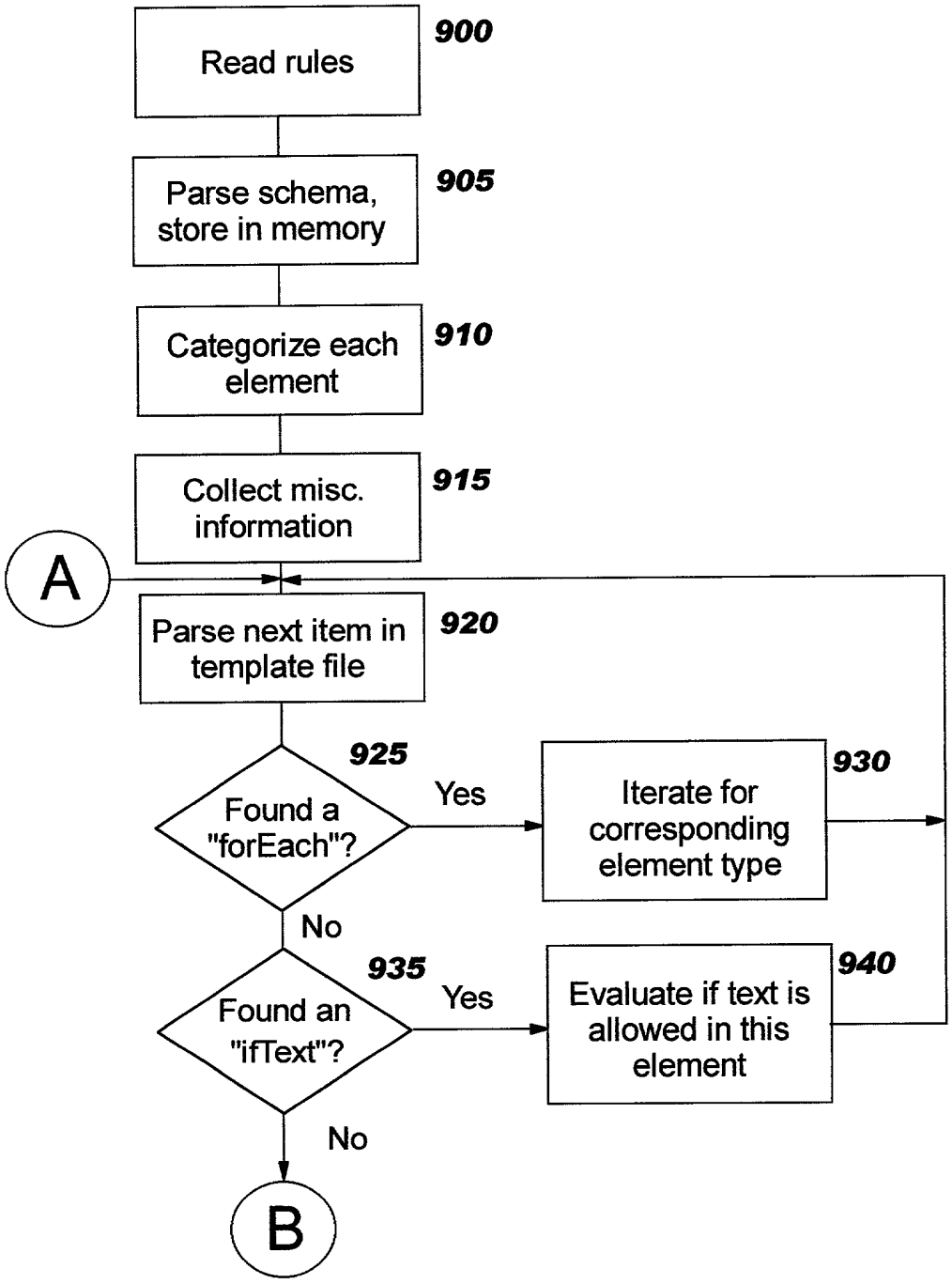

Read rules
900
Parse schema, store in memory
905
Categorize each element
910
Collect misc. information
915
A
Parse next item in template file
920
925
Found a "forEach"?
Yes
Iterate for corresponding element type
930
No
935
Found an "ifText"?
Yes
Evaluate if text is allowed in this element
940
No
B

FIG. 9B

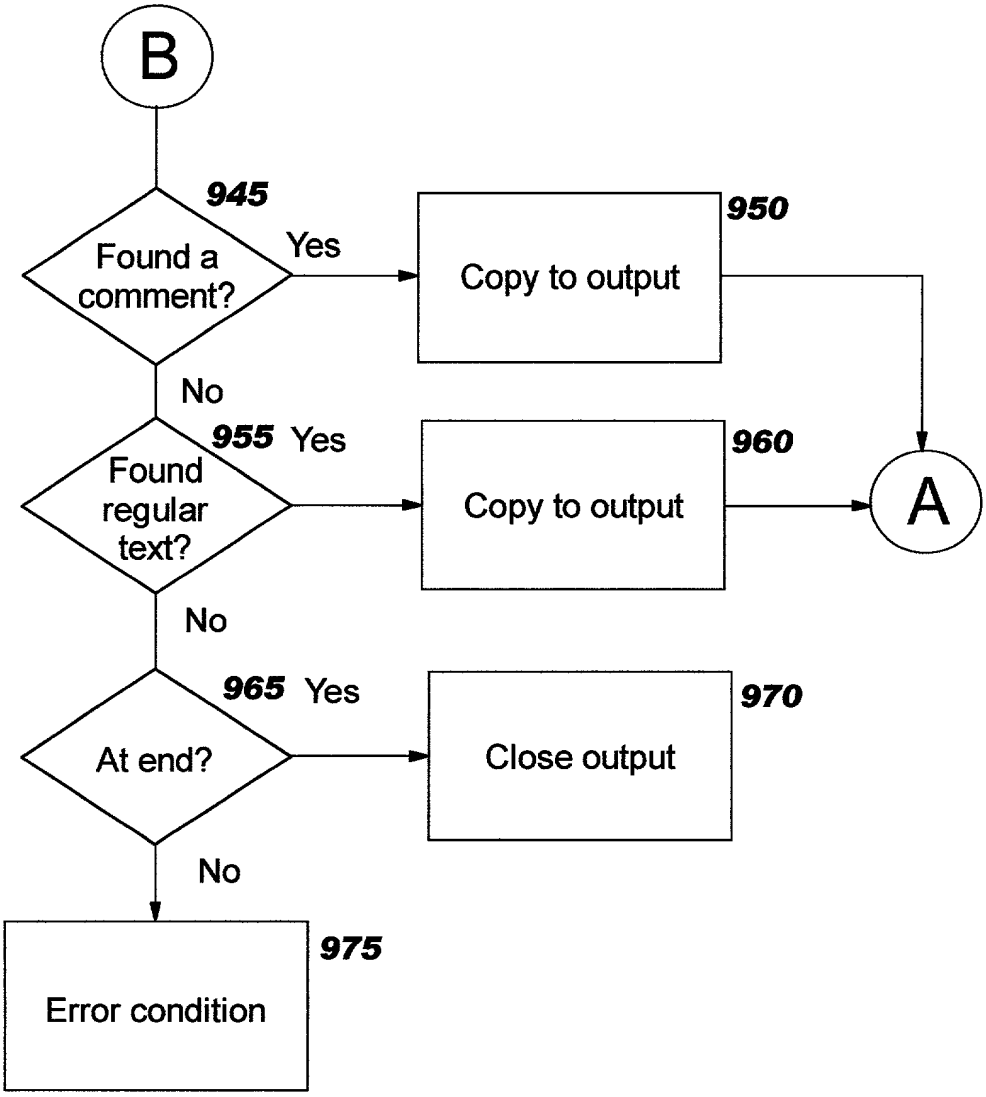
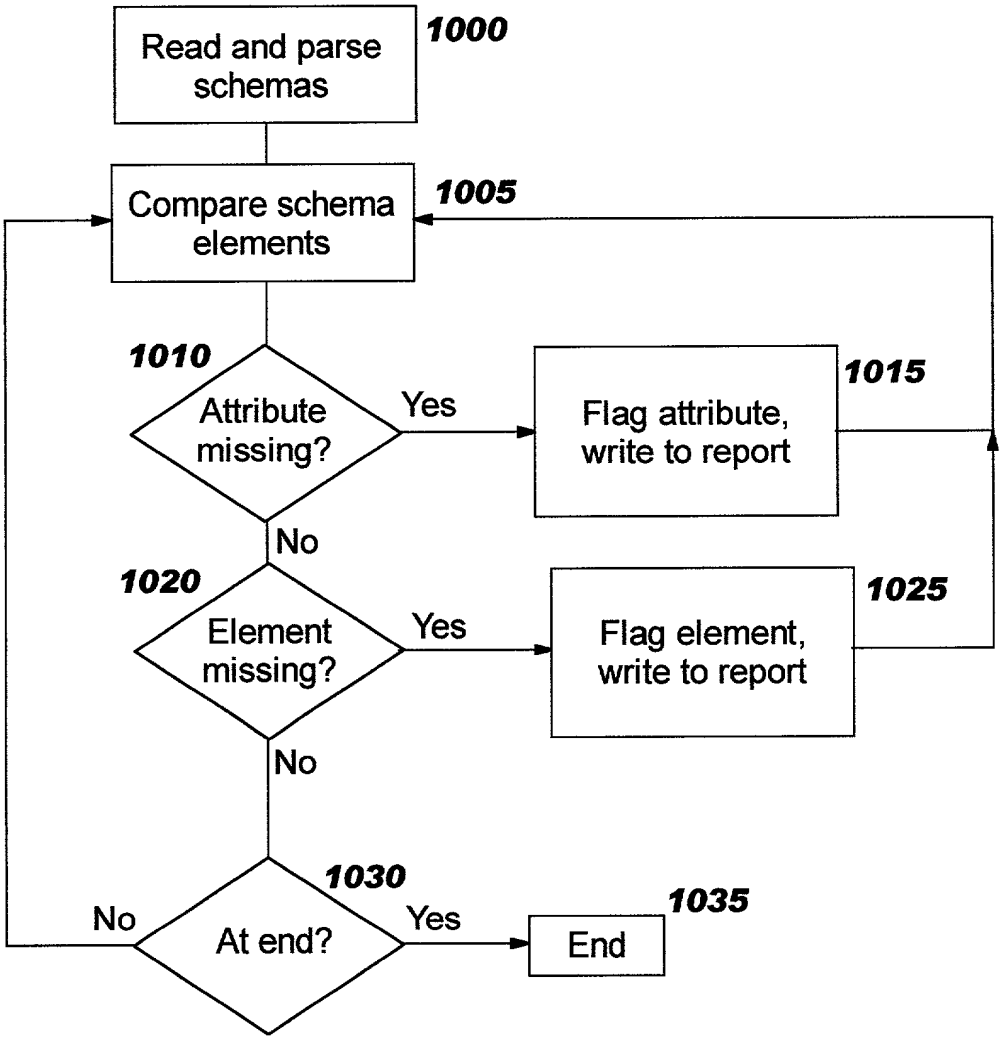

B
945
Found a comment?
Yes
950
Copy to output
No
955
Found regular text?
Yes
960
Copy to output
A
No
965
At end?
Yes
970
Close output
No
975
Error condition

FIG. 10


Read and parse schemas
1000
Compare schema elements
1005
1010
Attribute missing?
Yes
Flag attribute, write to report
1015
No
1020
Element missing?
Yes
Flag element, write to report
1025
No
1030
At end?
No
Yes
End
1035